Null Pointers

Daniel Plakosh, Software Engineering Institute [vita[1]]

2005-09-27 L3 / D/P, L[2]

One obvious technique to reduce vulnerabilities in C and C++ programs is to set the pointer to null after the call to `free()` has completed.

Development Context

Dynamic memory management

Technology Context

C, UNIX, Win32

Attacks

Attacker executes arbitrary code on machine with permissions of compromised process or changes the behavior of the program.

Risk

Standard C dynamic memory management functions such as `malloc()` , `calloc()` , realloc(), and `free()` [ISO/IEC 99] are prone to programmer mistakes that can lead to vulnerabilities resulting from buffer overflow in the heap, writing to already freed memory, and freeing the same memory multiple times (e.g., double-free vulnerabilities).

Description

One obvious technique to reduce vulnerabilities in C and C++ programs is to set the pointer to null after the call to `free()` has completed. Dangling pointers (pointers to already freed memory) can result in writing to freed memory and double-free vulnerabilities. Any attempt to dereference the pointer will result in a fault, which increases the likelihood that the error will be detected during implementation and test. Also, if the pointer is set to null, the memory can be freed multiple times without consequence.

While setting the pointer to null should significantly reduce vulnerabilities resulting from writing to freed memory and double-free vulnerabilities, it cannot prevent them when multiple pointers all reference the same data structure. Unfortunately, memory management in C and C++ must be performed with great care.

References

[ISO/IEC 99]	ISO/IEC. *ISO/IEC 9899 Second edition 1999-12-01 Programming languages — C*. International Organization for Standardization, 1999.

Pearson Education, Inc. Copyright

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/268-BSI.html (Plakosh, Daniel)